6. 순환 신경망 이해

- 순환 신경망에 대해 이해한다.
- 순환 신경망의 오차역전파를 이해한다.
- 순환 신경망을 구현하여 텍스트를 분류한다.
- 케라스를 이용하여 순환 신경망을 구현한다.

6. 순환 신경망 이해

순차 데이터 소개

: 우리가 다루는 데이터 중에는 독립적이지 않고 샘플이 서로 연관되어 있는 경우가 많다.

예) 날씨 -> 오후 3시의 온도를 알고 있다면 1시간 후의 온도를 비슷하게 예상할 수 있다.
즉, 온도를 매시간 측정하여 데이터 세트를 만들었다면 각 시간의 온도는 이전 시간의 온도와 깊은 연관이 있다.

이렇게 일정 시간 간격으로 배치된 데이터를 시계열(time series) 데이터라 한다.

순차 데이터 : 시계열 데이터를 포함하여 샘플에 순서가 있는 데이터를 일반적으로 순차 데이터(sequential data)라 한다.

텍스트 :
- 대표적인 순차 데이터
- 글을 구성하는 글자와 단어들의 순서가 맞아야 의미가 제대로 전달되기 때문
- 모델에서 순차 데이터를 처리하는 각 단계를 타임 스텝(time step)이라 한다.

Hello Deep Learning

다음은 3개의 단어로 이루어진 순차 데이터이다.

데이터의 처리 단위가 단어라면 총 타임 스텝은 3이다.

단어가 순서를 바꾸어 나열될 경우의 수는 3! 인 6이다.

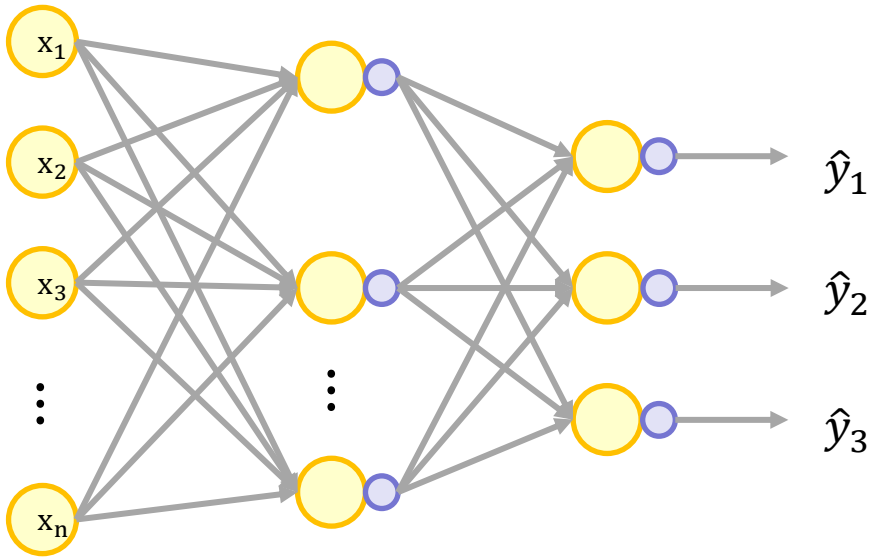
완전 신경망에 순차데이터를 주입

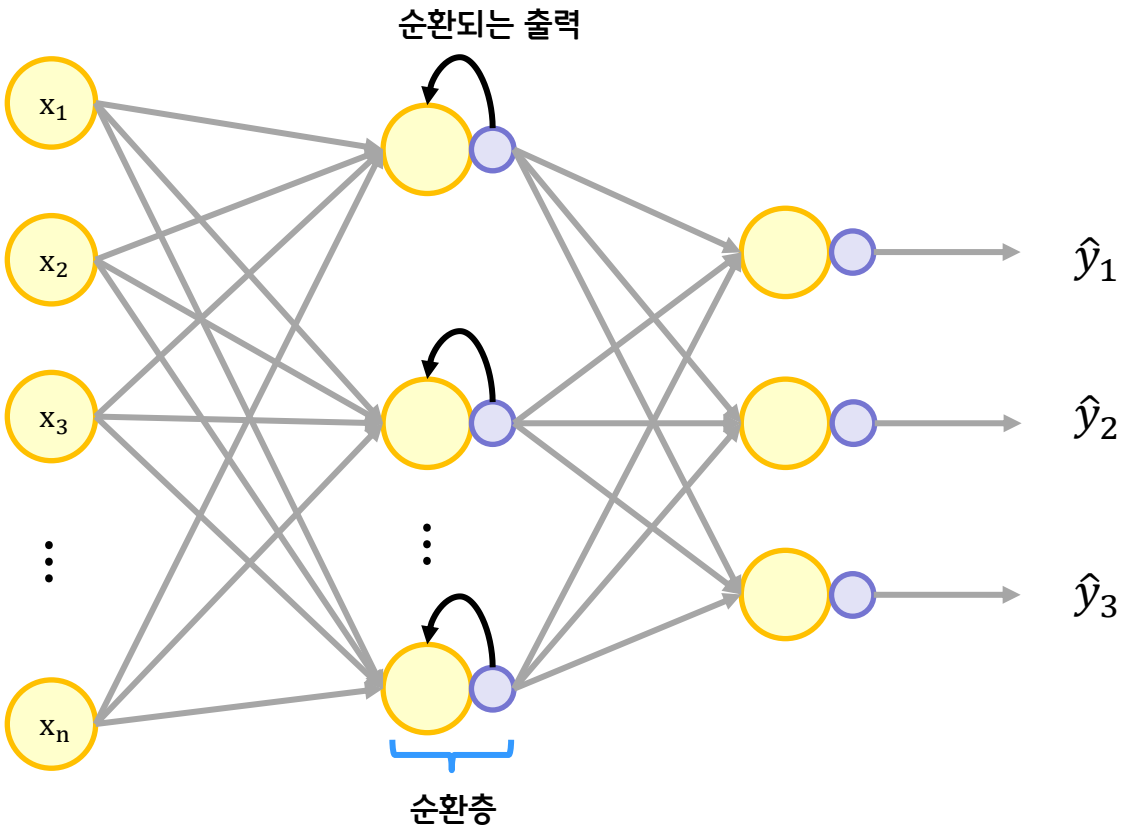
샘플을 무작위로 주입한다.

Learning Deep Hello

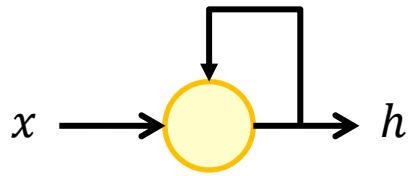
Hello Deep Learning

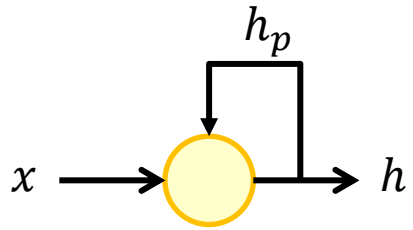
Deep Hello Learning

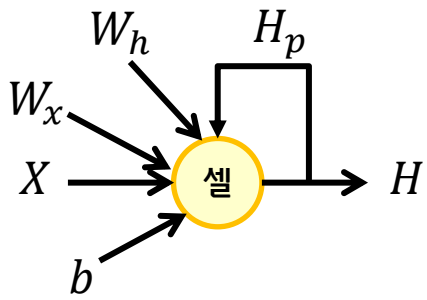



순환되는 출력
x_1
x_2
x_3
x_n
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
순환층

순환 신경망은 뉴런을 셀이라 부른다.



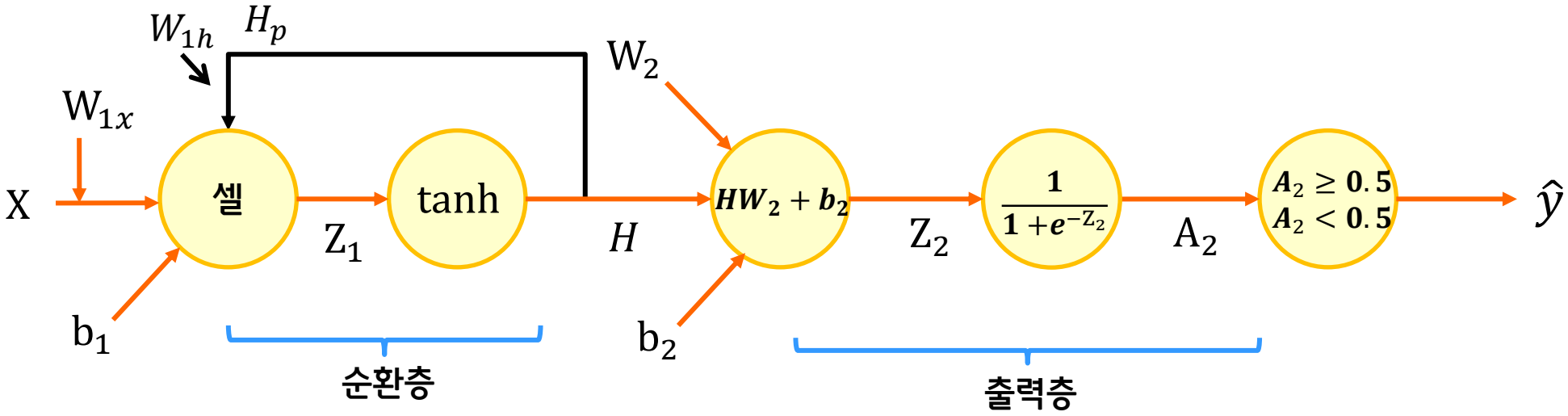




$$Z = XW_x + H_pW_h + b$$

$$(H = \tanh(Z))$$

순환 신경망의 정방향 계산



순환층의 정방향 계산	출력층의 정방향 계산
$Z_1 = XW_{1x} + H_pW_{1h} + b_1$ $H = \tanh(Z_1)$	$Z_2 = HW_2 + b_2$ $A_2 = sigmoid(Z_2)$

정방향 계산에 필요한 입력과 가중치의 구조

입력 데이터 X의 크기는 (m, n_f) 이다. m 은 샘플 개수 이고 n_f는 특성 개수 이다.

$$X = \begin{bmatrix} \cdots \\ \vdots \quad \ddots \quad \vdots \\ \cdots \end{bmatrix} \begin{matrix} m \text{ (샘플 개수)} \\ \\ \\ n_f \\ \text{(특성 개수)} \end{matrix}$$

입력에 곱해지는 가중치 W_{1x} 의 크기는 (n_f, n_c) 이다. n_c는 순환층에 있는 셀의 개수 이다.

$$W_{1x} = \begin{bmatrix} \cdots \\ \vdots \quad \ddots \quad \vdots \\ \cdots \end{bmatrix} \begin{matrix} n_f \\ \\ \\ n_c \\ \text{(순환층의 뉴런 개수)} \end{matrix}$$

입력 데이터 X와 가중치 W_{1x} 에 점 곱 연산을 적용한 결과의 크기는 (m, n_c) 이다.

$$XW_{1x} = (m, n_f) \cdot (n_f, n_c) = (m, n_c)$$

XW_{1x} 의 크기가 (m, n_c) 이므로, Z_1과 H 그리고 이전 은닉상태인 H_p 도 (m, n_c) 가 된다.
두 행렬을 곱한 H_pW_{1h} 크기도 (m, n_c) 가 된다.
이를 통해 H_p 와 곱해지는 가중치 W_{1h} 의 크기가 (n_c, n_c) 가 된다는 것을 알 수 있다.

$$H_pW_{1h} = (m, n_c) \cdot (n_c, n_c) = (m, n_c)$$

출력층으로 전달되는 H의 크기가 (m, n_c) 이므로 이와 곱해지는 가중치 W_2 의 크기는 $(n_c, n\_classes)$ 이다.
따라서 Z_2와 A_2 의 크기는 $(n_c, n\_classes)$ 이다.
이진 분류를 다루므로 $n_classes$는 1이다.

$$HW_2 = (m, n_c) \cdot (n_c, 1) = (m, 1)$$

각 층의 뉴런마다 절편이 하나씩 필요하므로 b_1의 크기는 $(n_c, 1)$ 이고 b_2의 크기는 $(n_classes,)$이다.

6. 순환 신경망 이해

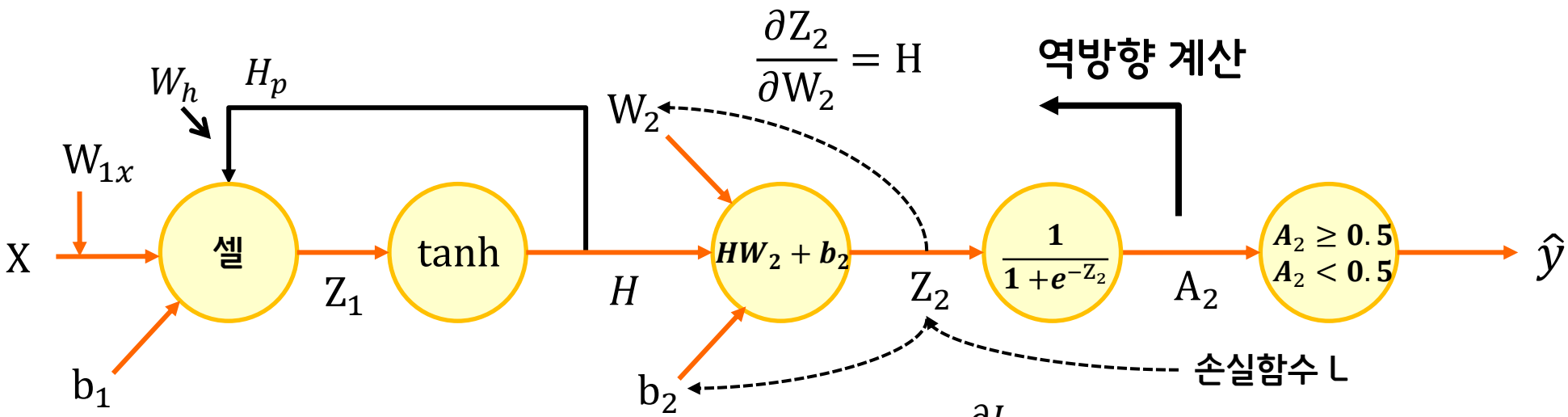
6.1 순환 신경망

6.2 순환 신경망의 오차역전파

6.3 순환 신경망 구현

6.4 케라스를 이용한 순환 신경망 구현

가중치 W_2에 대한 손실 함수의 도함수를 구한다.



$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = H^T(A_2 - Y) \qquad (m, n_c)^T(m, 1)$$

$$(n_c, m)(m, 1)$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial b_2} = 1^T(A_2 - Y) \Rightarrow sum(A_2 - Y)$$

H에 대한 Z_2의 도함수를 구한다.

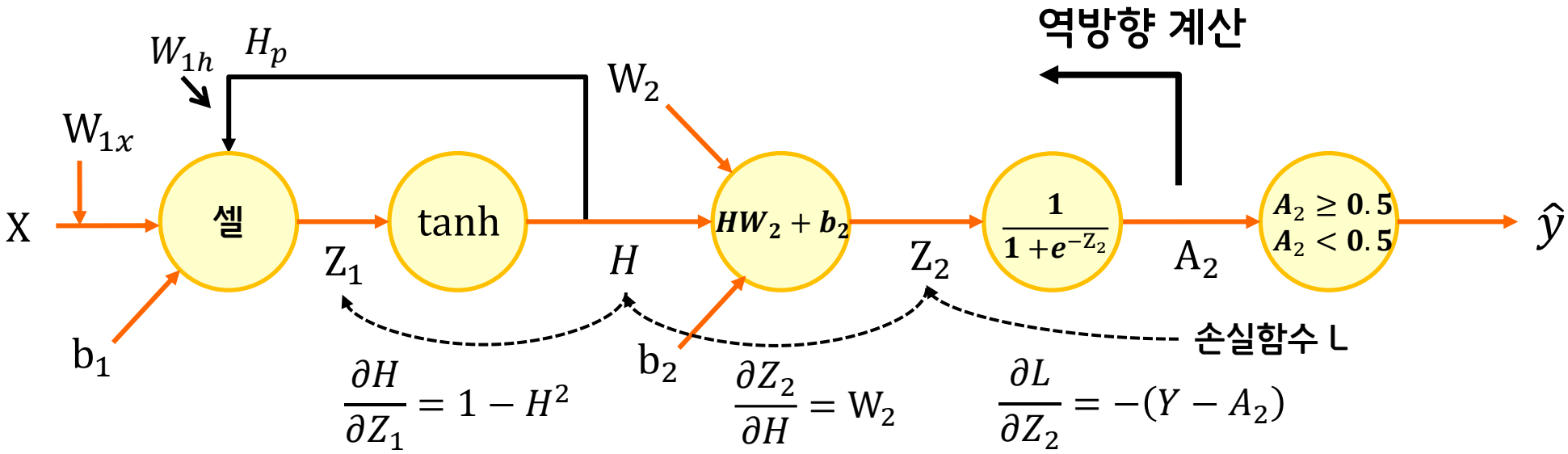
$$\frac{\partial Z_2}{\partial H} = \frac{\partial}{\partial H}(H\mathrm{W}_2 + b_2) = \mathrm{W}_2$$

Z_1에 대한 H의 도함수를 구한다.

$$\frac{\partial H}{\partial Z_1} = \frac{\partial}{\partial Z_1}\tanh(Z_1) = 1 - tanh^2(Z_1) = 1 - H^2$$

$$(1 - \tanh(Z_1))(1 + \tanh(Z_1))$$

Z_1에 대한 손실 함수의 도함수를 구한다.



$$\frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial H}\frac{\partial H}{\partial Z_1} = (A_2 - Y)W_2^T \odot (1 - H^2)$$

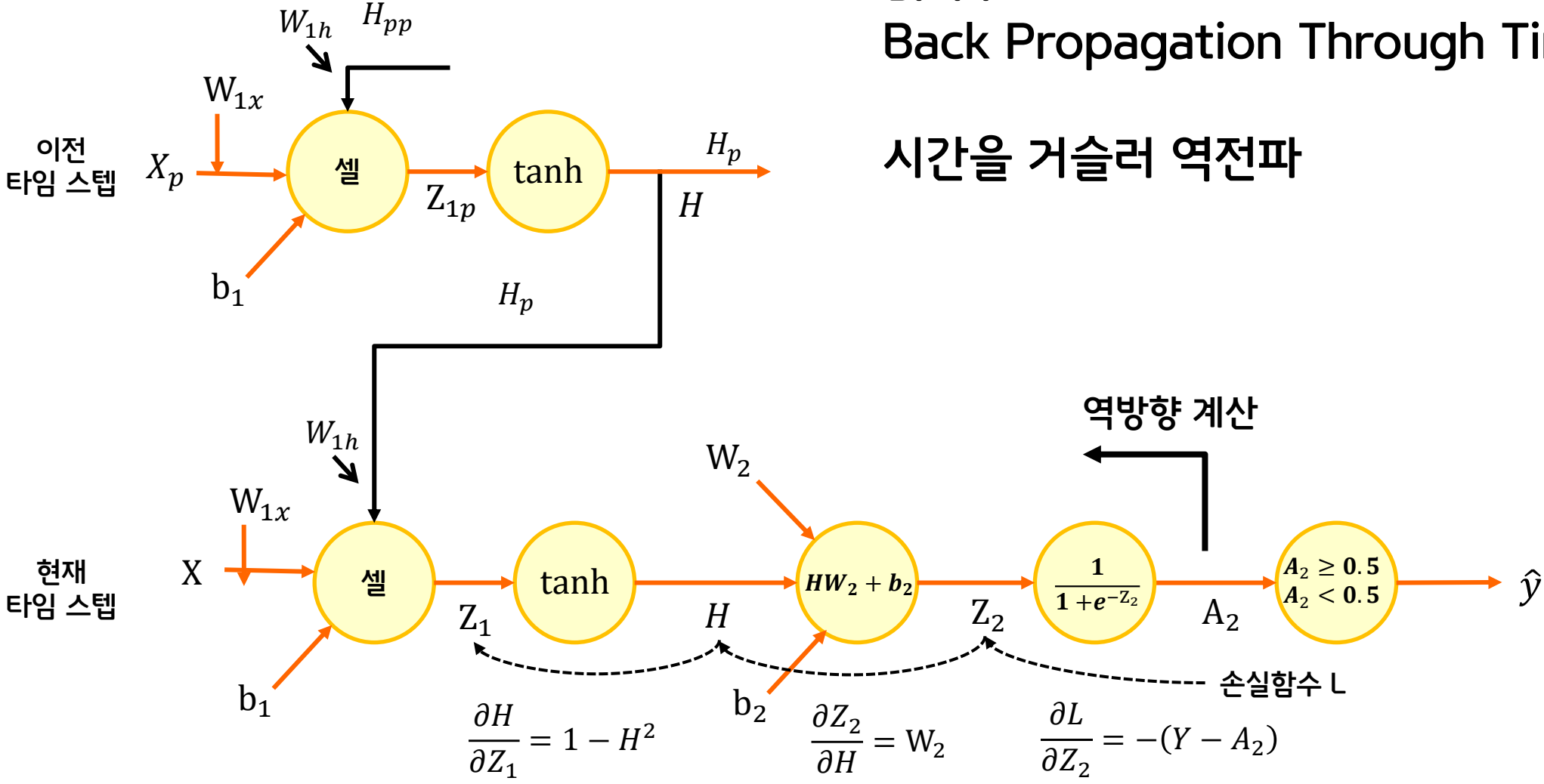
$$(m, 1)(n_c, 1)^T$$
$$(m, 1)(1, n_c)$$

가중치 W_{1h}에 대한 Z_1의 도함수를 구한다.

$$\frac{\partial Z_1}{\partial W_{1h}} = \frac{\partial}{\partial W_{1h}}\left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right) = H_p$$

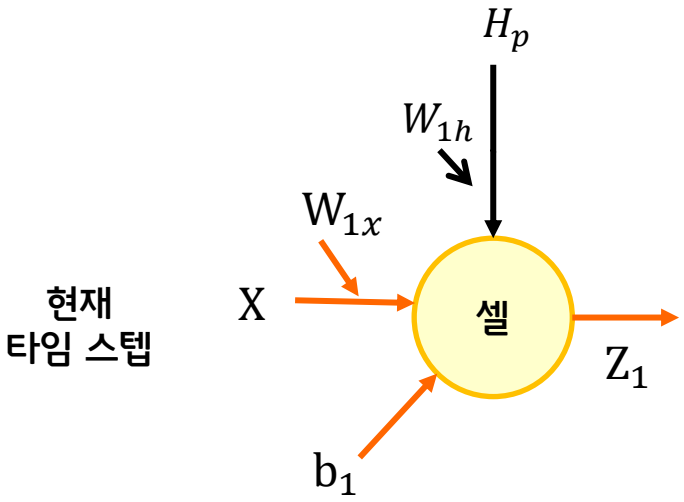
BPTT:
Back Propagation Through Time

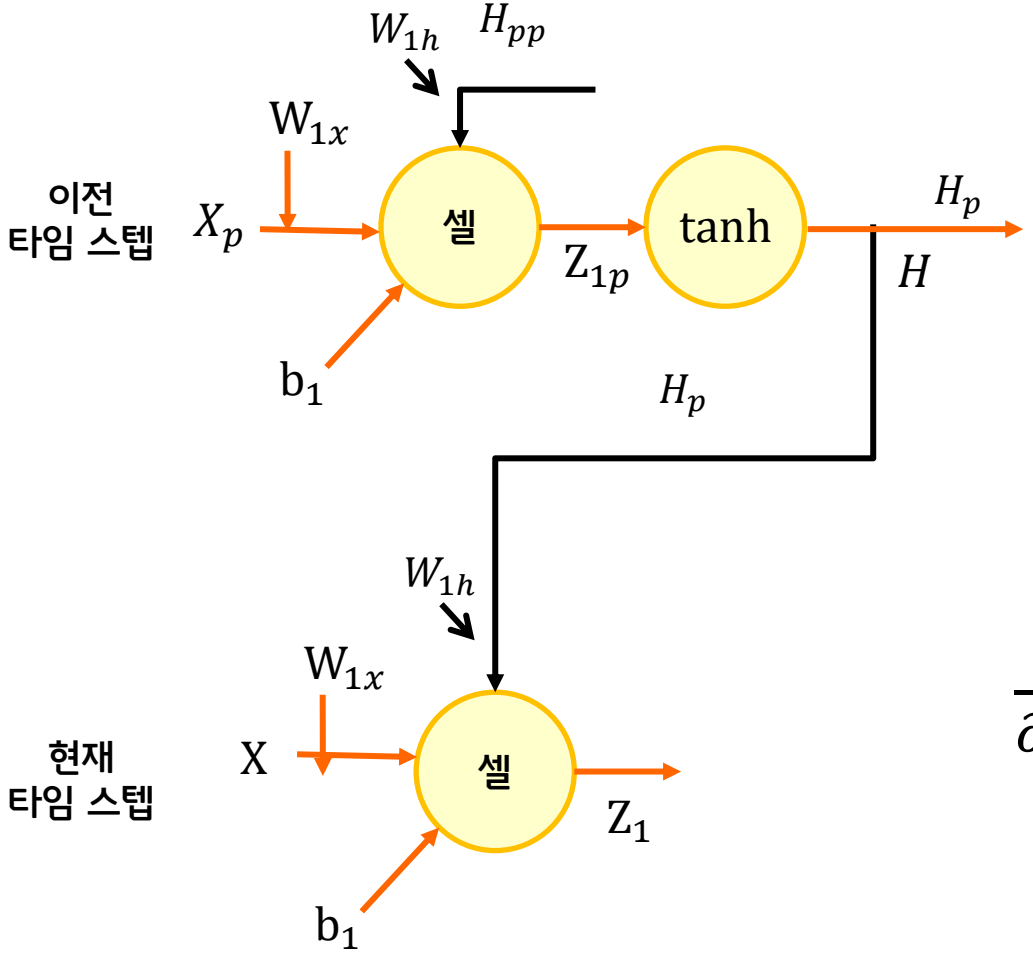
시간을 거슬러 역전파



$$\frac{\partial Z_1}{\partial W_{1h}} = \frac{\partial}{\partial W_{1h}}\left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right)$$

$$\frac{\partial Z_1}{\partial W_{1h}} = H_p$$





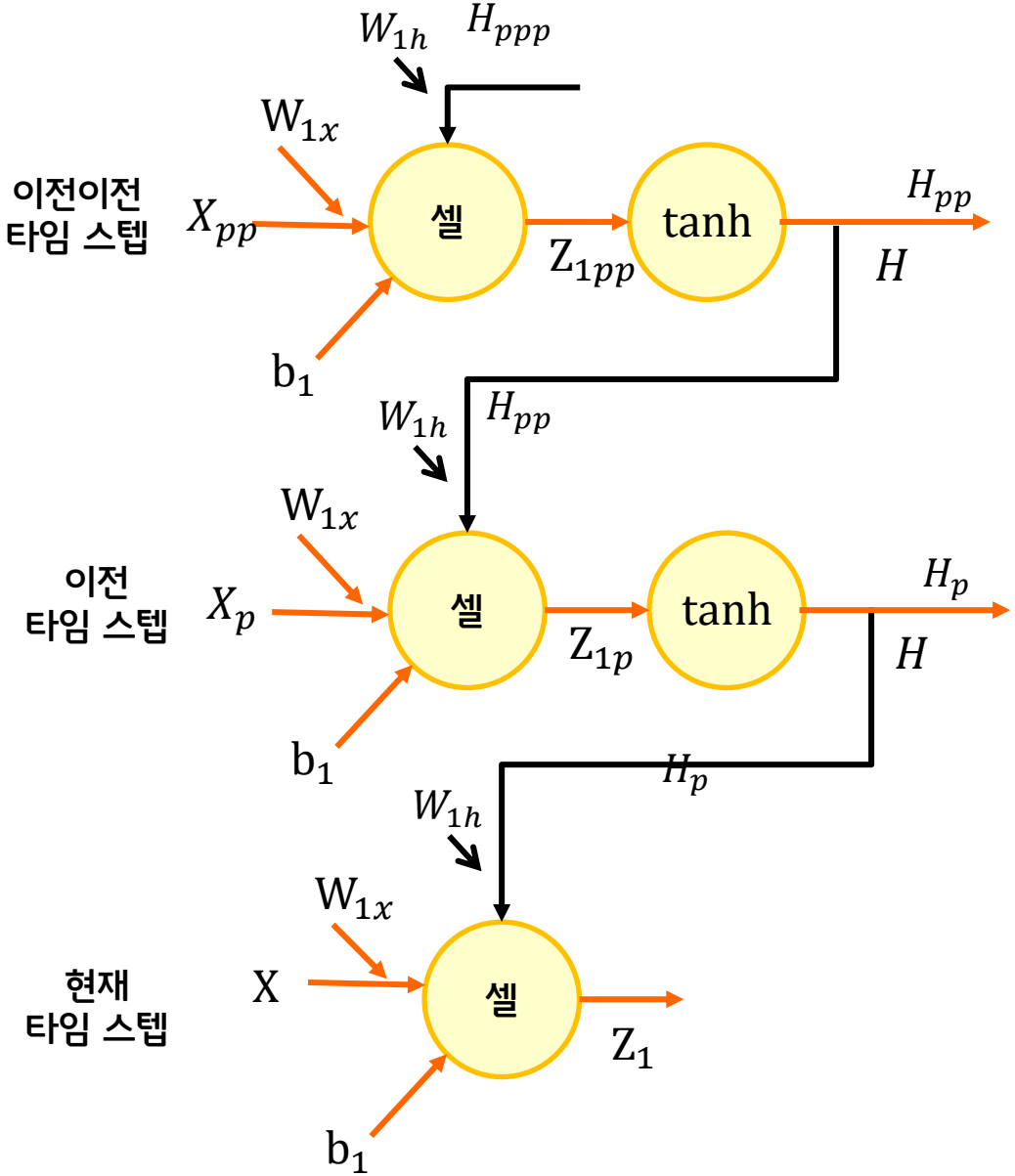
$$Z_{1p} = \left(X_p \mathrm{W}_{1x} + H_{pp} \mathrm{W}_{1h} + b_1\right)$$

$$H_p = \tanh(Z_{1p})$$

$$Z_1 = \left(X \mathrm{W}_{1x} + H_p \mathrm{W}_{1h} + b_1\right)$$

$$\frac{\partial Z_1}{\partial W_{1h}} = \ldots + \frac{\partial Z_1}{\partial H_p} \frac{\partial H_p}{\partial Z_{1p}} \frac{\partial Z_{1p}}{\partial W_{1h}}$$

$$\frac{\partial Z_1}{\partial W_{1h}} = H_p + H_{pp} W_{1h} \odot (1 - H_p^2)$$



$$Z_{1pp} = \left(X_{pp}\mathrm{W}_{1x} + H_{ppp}\mathrm{W}_{1h} + b_1\right)$$
$$H_{pp} = \tanh(Z_{1pp})$$
$$Z_{1p} = \left(X_p\mathrm{W}_{1x} + H_{pp}\mathrm{W}_{1h} + b_1\right)$$
$$H_p = \tanh(Z_{1p})$$
$$Z_1 = \left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right)$$

$$\frac{\partial Z_1}{\partial W_{1h}} = \cdots + \frac{\partial Z_1}{\partial H_p}\frac{\partial H_p}{\partial Z_{1p}}\frac{\partial Z_{1p}}{\partial H_{pp}}\frac{\partial H_{pp}}{\partial Z_{1pp}}\frac{\partial Z_{1pp}}{\partial W}$$

$$\frac{\partial Z_1}{\partial W_{1h}} = H_p + H_{pp}W_{1h}\odot\left(1 - H_p^2\right) +$$

$$H_{ppp}W_{1h}\odot\left(1 - H_p^2\right)W_{1h}\odot\left(1 - H_{pp}^2\right)$$

가중치 W_{1x}에 대한 Z_1의 도함수를 구한다.

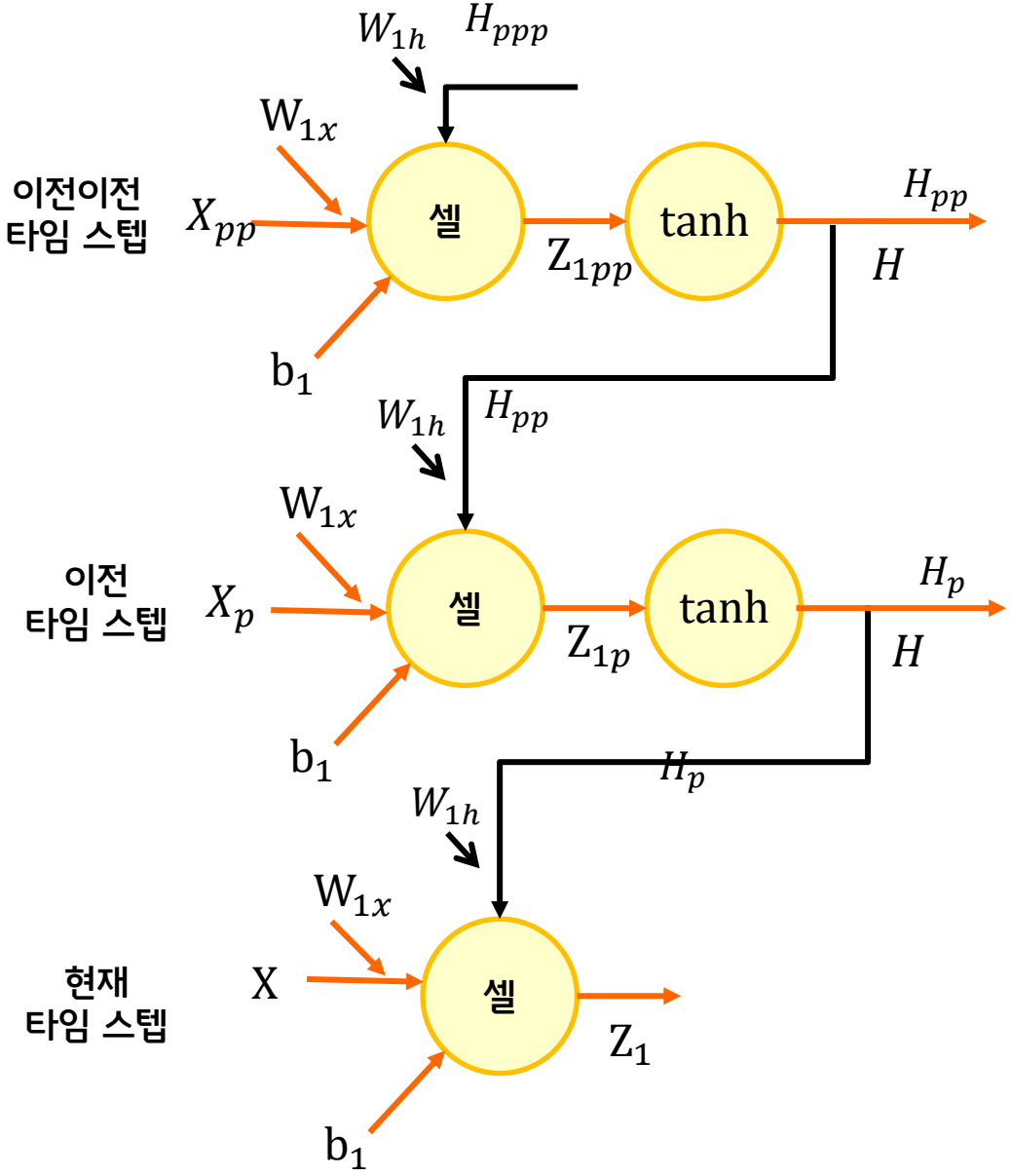
$$\frac{\partial Z_1}{\partial W_{1x}} = \frac{\partial}{\partial W_{1x}}\left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right)$$

TBPTT:
Truncate Back Propagation Th

특정 범위 안에서 시간을 거슬러 역

$$\frac{\partial Z_1}{\partial W_{1x}} = X + XW_{1h}\odot\left(1 - H_p^2\right) + XW_{1h}\odot\left(1 - H_p^2\right)\odot W_{1h}\odot\left(1 - H_{pp}^2\right) + \cdots$$

정리된 식을 보면 가중치 W_{1h}와 tanh함수의 도함수인 $1 - H^2$이 계속 곱해지면서
더해지는 것을 볼 수 있다.



$$Z_{1pp} = \left(X_{pp}\mathrm{W}_{1x} + H_{ppp}\mathrm{W}_{1h} + b_1\right)$$
$$H_{pp} = \tanh(Z_{1pp})$$
$$Z_{1p} = \left(X_p\mathrm{W}_{1x} + H_{pp}\mathrm{W}_{1h} + b_1\right)$$
$$H_p = \tanh(Z_{1p})$$
$$Z_1 = \left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right)$$

$$\frac{\partial Z_1}{\partial W_{1x}} = \frac{\partial Z_1}{\partial H_p}\frac{\partial H_p}{\partial Z_{1p}}\frac{\partial Z_{1p}}{\partial H_{pp}}\frac{\partial H_{pp}}{\partial Z_{1pp}}\frac{\partial Z_{1pp}}{\partial W_{1x}}$$

$$\frac{\partial Z_1}{\partial W_{1x}} = X + X_p W_{1h}\odot\left(1 - H_p^2\right) +$$

$$X_{pp}W_{1h}\odot\left(1 - H_p^2\right)W_{1h}\odot\left(1 - H_{pp}^2\right)$$

가중치 W_{1h}에 대한 Z_1의 도함수를 구한다.

$$\frac{\partial Z_1}{\partial W_{1h}} = \frac{\partial}{\partial W_{1h}}\left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right) = H_p$$

$$H_p = X_p\mathrm{W}_{1x} + H_{pp}\mathrm{W}_{1h} + b_1$$

$$Z = X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1$$

H_p 도 W_{1h}을 사용하므로 위처럼 상수로 처리 하면 안된다.

$$\frac{\partial Z_1}{\partial W_{1h}} = \frac{\partial Z_1}{\partial H_p}\frac{\partial H_p}{\partial W1h}$$

$$\frac{\partial Z_1}{\partial W_{1h}} = H_p + H_{pp}W_{1h}\odot\left(1 - H_p^2\right) + H_{ppp}W_{1h}\odot\left(1 - H_p^2\right)\odot W_{1h}\odot\left(1 - H_{pp}^2\right) + \cdots$$

순환 신경망의 역전파을 구현하려면 각 타임 스텝마다 셀의 출력을 모두 기록해서 가지고 있어야 한다. $(H_p, H_{pp}, H_{ppp}, \cdots)$

정리된 식을 보면 가중치 W_{1h}와 tanh함수의 도함수인 $1 - H^2$이 계속 곱해지면서 더해지는 것을 볼 수 있다.

가중치 b_1에 대한 Z_1의 도함수를 구한다.

$$\frac{\partial Z_1}{\partial b_1} = \frac{\partial}{\partial b_1}\left(X\mathrm{W}_{1x} + H_p\mathrm{W}_{1h} + b_1\right)$$

$$\frac{\partial Z_1}{\partial b_1} = 1 + W_{1h}\odot\left(1 - H_p^2\right) + W_{1h}\odot\left(1 - H_p^2\right)\odot W_{1h}\odot\left(1 - H_{pp}^2\right) + \cdots$$

구현 시 $\frac{\partial L}{\partial Z_1}$를 err_to_cell이라 표현하고

최종 그레이디언트는 err_to_cell과 $\frac{\partial Z_1}{\partial W_{1h}}\frac{\partial Z_1}{\partial W_x}\frac{\partial Z_1}{\partial b_1}$를 곱해서 구한다.

6. 순환 신경망 이해

6.1 순환 신경망

6.2 순환 신경망의 오차역전파

6.3 순환 신경망 구현

6.4 케라스를 이용한 순환 신경망 구현

훈련 세트와 검증 세트를 준비

1. 텐서프로에서 IMDB 데이터 세트 불러오기

```
import numpy as np
from tensorflow.keras.datasets import imdb
(x_train_all, y_train_all), (x_test, y_test) = imdb.load_data(skip_top=20, num_words=100)
```

2. 훈련 세트 크기 확인

```
print(x_train_all.shape, y_train_all.shape)
```

```
(25000,) (25000,)
```

3. 훈련 세트 샘플 확인

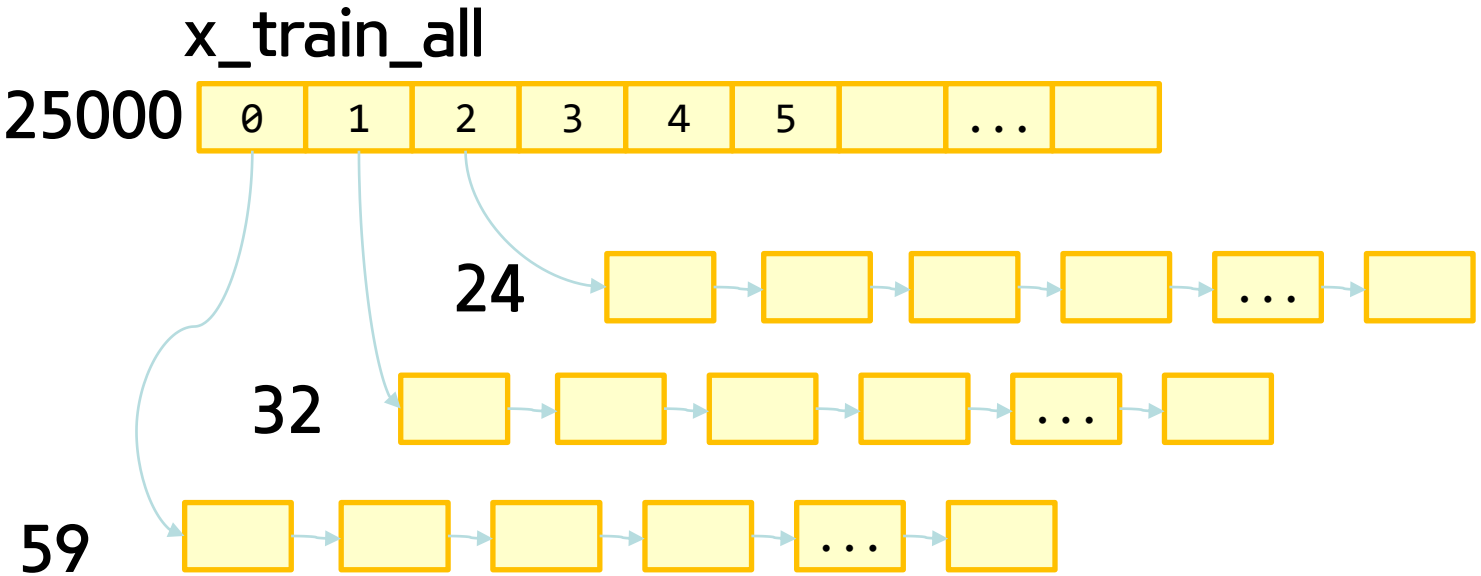
```
print(x_train_all[0])
```

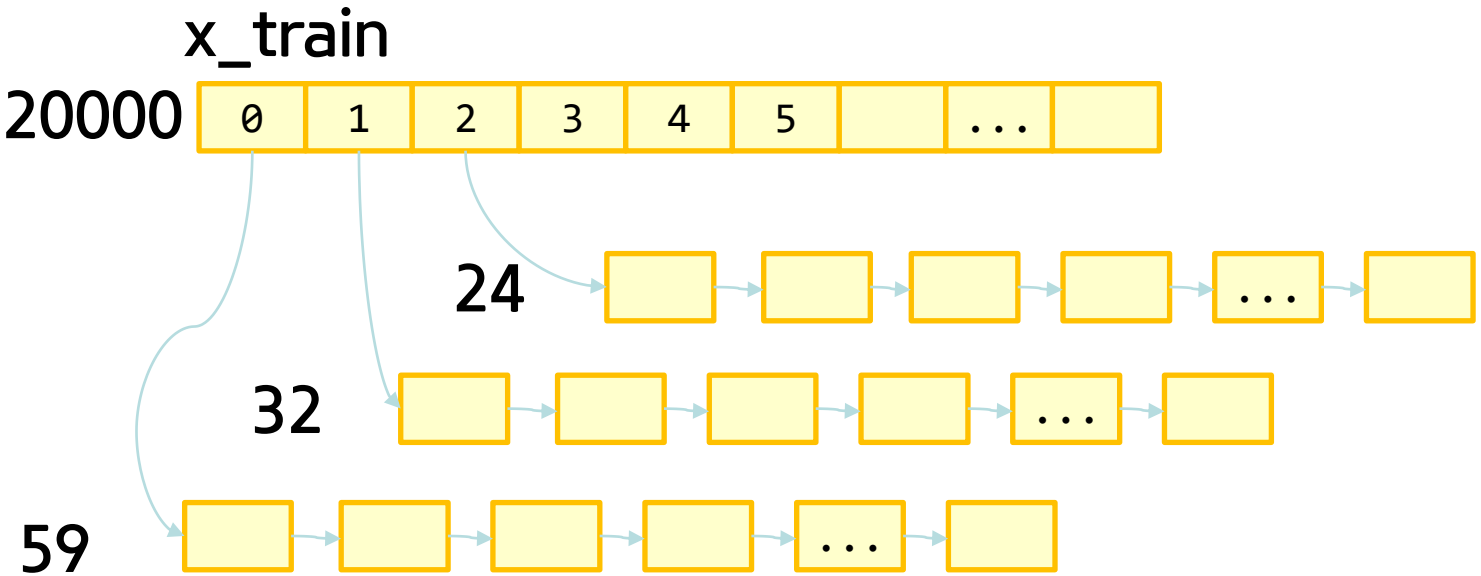
```
[2, 2, 22, 2, 43, 2, 2, 2, 2, 65, 2, 2, 66, 2, 2, 2, 36, 2, 2, 25, 2, 43, 2, 2, 50, 2, 2, 2, 35, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 39, 2, 2, 2, 2, 2, 2, 38, 2, 2, 2, 2, 50, 2, 2, 2, 2, 2, 2, 22, 2, 2, 2, 2,
2, 22, 71, 87, 2, 2, 43, 2, 38, 76, 2, 2, 2, 2, 22, 2, 2, 2, 2, 2, 2, 2, 2, 2, 62, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 66, 2, 33, 2, 2, 2, 2, 38, 2, 2, 25, 2, 51, 36, 2, 48, 25, 2, 33, 2, 22, 2, 2, 28, 77,
52, 2, 2, 2, 2, 82, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 36, 71, 43, 2, 2, 26, 2, 2, 46, 2, 2, 2,
2, 2, 2, 88, 2, 2, 2, 2, 98, 32, 2, 56, 26, 2, 2, 2, 2, 2, 2, 2, 22, 21, 2, 2, 26, 2, 2, 2, 30, 2, 2,
51, 36, 28, 2, 92, 25, 2, 2, 2, 65, 2, 38, 2, 88, 2, 2, 2, 2, 2, 2, 2, 2, 32, 2, 2, 2, 2, 2, 32]
```

4. 훈련 세트에서 2 제외

```
for i in range(len(x_train_all)):
    x_train_all[i] = [w for w in x_train_all[i] if w > 2]
print(x_train_all[0])
```

```
[22, 43, 65, 66, 36, 25, 43, 50, 35, 39, 38, 50, 22, 22, 71, 87, 43, 38, 76, 22, 62, 66, 33, 38, 25, 51,
36, 48, 25, 33, 22, 28, 77, 52, 82, 36, 71, 43, 26, 46, 88, 98, 32, 56, 26, 22, 21, 26, 30, 51, 36, 28,
92, 25, 65, 38, 88, 32, 32]
```


x_train_all
25000
0
1
2
3
4
5
...
24
32
59


x_train
20000
0
1
2
3
4
5
...
24
...
32
...
59
...

5. 어휘 사전 내려 받기

```
word_to_index = imdb.get_word_index()
word_to_index['movie']
```

```
17
```

6. 훈련 세트의 정수를 영단어로 변환

```
index_to_word = {word_to_index[k]: k for k in word_to_index}

for w in x_train_all[0]:
    print(index_to_word[w - 3], end=' ')
```

```
film just story really they you just there an from so there film film were great just so much film
would really at so you what they if you at film have been good also they were just are out because them
all up are film but are be what they have don't you story so because all all
```

7. 훈련 샘플의 길이 확인

```
print(len(x_train_all[0]), len(x_train_all[1]))
```

```
218 189
```

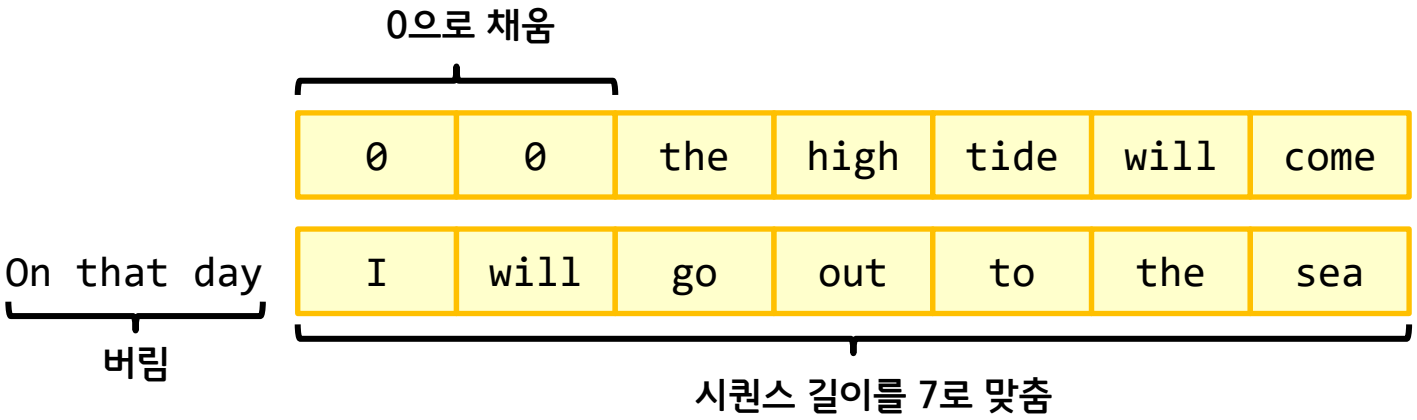
8. 훈련 세트의 타깃 데이터 확인

```
print(y_train_all[:10])
```

```
[1 0 0 1 0 0 1 0 1 0]
```

9. 검증 세트 준비

```
np.random.seed(42)
random_index = np.random.permutation(25000)

x_train = x_train_all[random_index[:20000]]
y_train = y_train_all[random_index[:20000]]
x_val = x_train_all[random_index[20000:]]
y_val = y_train_all[random_index[20000:]]
```



1. 텐서플로로 샘플의 길이 맞추기

```
from tensorflow.keras.preprocessing import sequence

maxlen=100
x_train_seq = sequence.pad_sequences(x_train, maxlen=maxlen)
x_val_seq = sequence.pad_sequences(x_val, maxlen=maxlen)
```

2. 길이를 조정한 훈련 세트의 크기와 샘플 확인

```
print(x_train_seq.shape, x_val_seq.shape)
```

```
(20000, 100) (5000, 100)
```

```
print(x_train_seq[0])
```

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0 35 40 27 28 40 22 83 31 85 45
 24 23 31 70 31 76 30 98 32 22 28 51 75 56 30 33 97 53 38 46 53 74 31 35
 23 34 22 58]
```

1. 텐서플로로 원-핫 인코딩

```
from tensorflow.keras.utils import to_categorical

x_train_onehot = to_categorical(x_train_seq)
x_val_onehot = to_categorical(x_val_seq)
```

2. 원-핫 인코딩으로 변환된 변수 x_train_onehot의 차원 크기 확인

```
print(x_train_onehot.shape)
```

```
(20000, 100, 100)
```

3. 원-핫 인코딩으로 변환된 변수 x_train_onehot의 byte 크기 확인

```
print(x_train_onehot.nbytes)
```

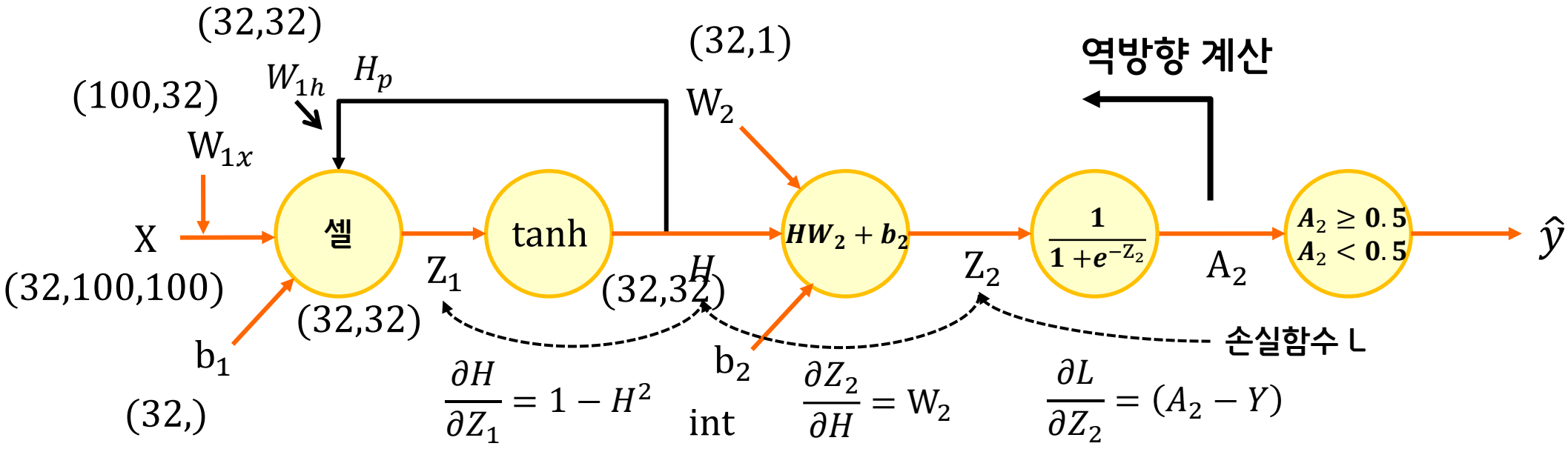
```
800000000
```

1. __init__() 함수 수정

```
def __init__(self, n_cells=10, batch_size=32, learning_rate=0.1):
        self.n_cells = n_cells      # 셀 개수
        self.batch_size = batch_size     # 배치 크기
        self.w1h = None             # 은닉 상태에 대한 가중치
        self.w1x = None             # 입력에 대한 가중치
        self.b1 = None              # 순환층의 절편
        self.w2 = None              # 출력층의 가중치
        self.b2 = None              # 출력층의 절편
        self.h = None               # 순환층의 활성화 출력
        self.losses = []            # 훈련 손실
        self.val_losses = []        # 검증 손실
        self.lr = learning_rate     # 학습률
```

2. 직교 행렬 방식으로 가중치 초기화

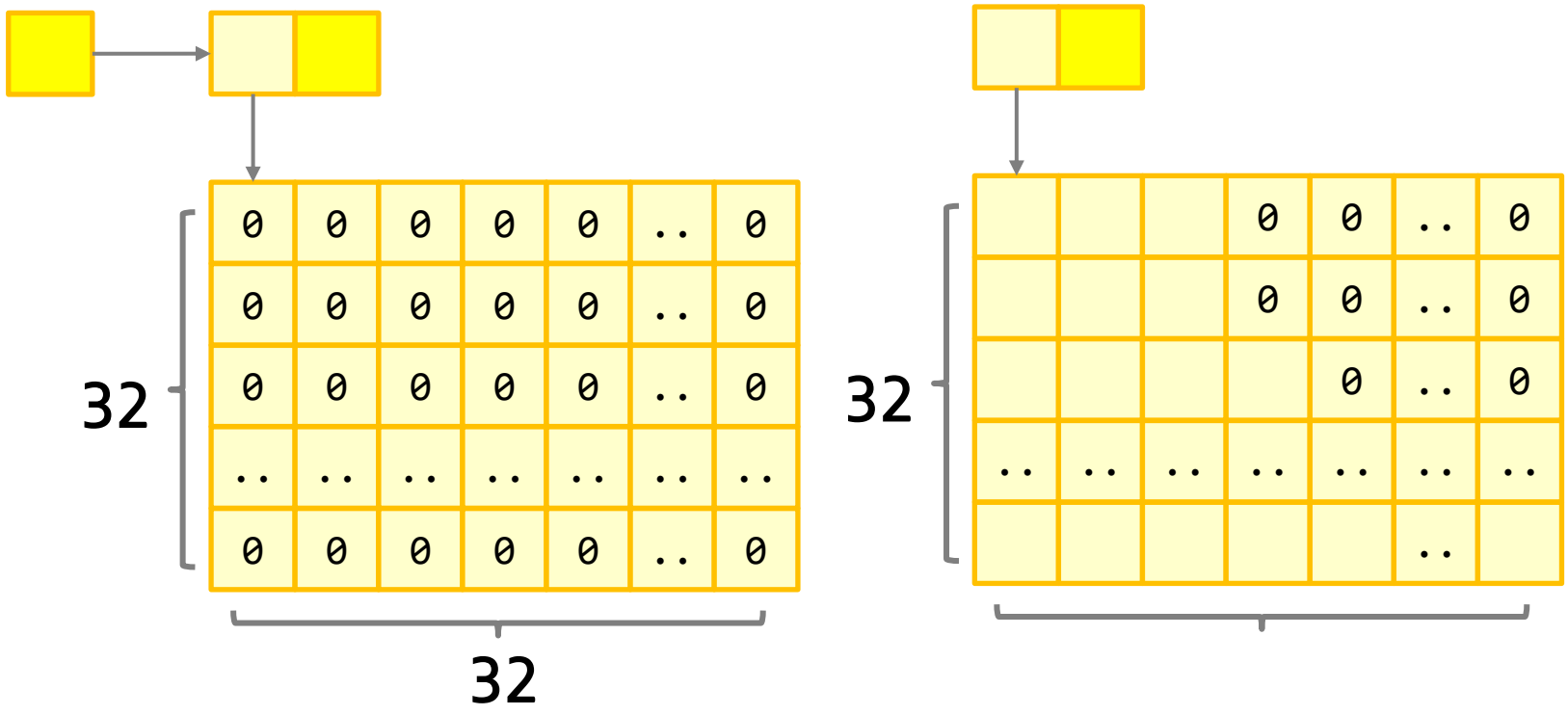
```
def init_weights(self, n_features, n_classes):
        orth_init = tf.initializers.Orthogonal()
        glorot_init = tf.initializers.GlorotUniform()

        self.w1h = orth_init((self.n_cells, self.n_cells)).numpy() # (셀 개수, 셀 개수)
        self.w1x = glorot_init((n_features, self.n_cells)).numpy() # (특성 개수, 셀 개수)
        self.b1 = np.zeros(self.n_cells)                           # 은닉층의 크기
        self.w2 = glorot_init((self.n_cells, n_classes)).numpy()   # (셀 개수, 클래스 개수)
        self.b2 = np.zeros(n_classes)
```


(32,32)
(32,1)
역방향 계산
(100,32)
W_{1h}
H_p
W_2
W_{1x}
X
(32,100,100)
셀
tanh
$HW_2 + b_2$
$\frac{1}{1+e^{-Z_2}}$
$A_2 \geq 0.5$
$A_2 < 0.5$
$\hat{y}$
Z_1
H
(32,32)
Z_2
A_2
(32,32)
손실함수 L
b_1
b_2
(32,)
int
$\frac{\partial H}{\partial Z_1} = 1 - H^2$
$\frac{\partial Z_2}{\partial H} = W_2$
$\frac{\partial L}{\partial Z_2} = (A_2 - Y)$

3. 정방향 계산 구현

```
def forpass(self, x):
        self.h = [np.zeros((x.shape[0], self.n_cells))]
```

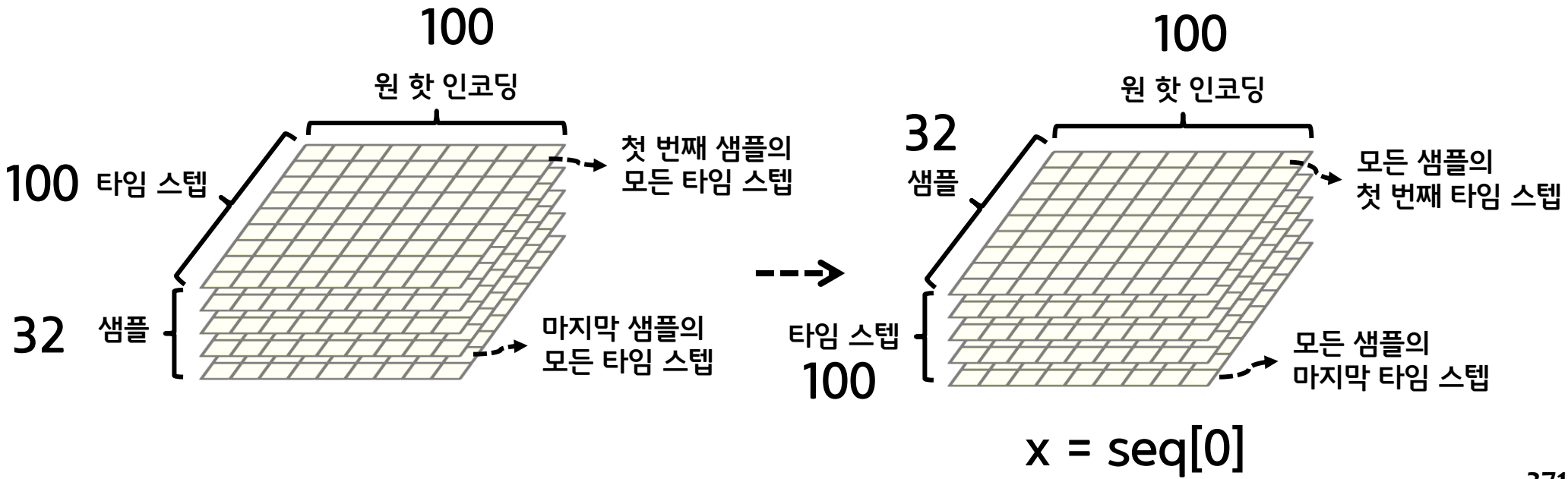
self.h



4. swapaxes() 함수를 사용하여 입력 x의 첫 번째 배치 차원과 두 번째 타임 스텝 차원을 바꾼다.

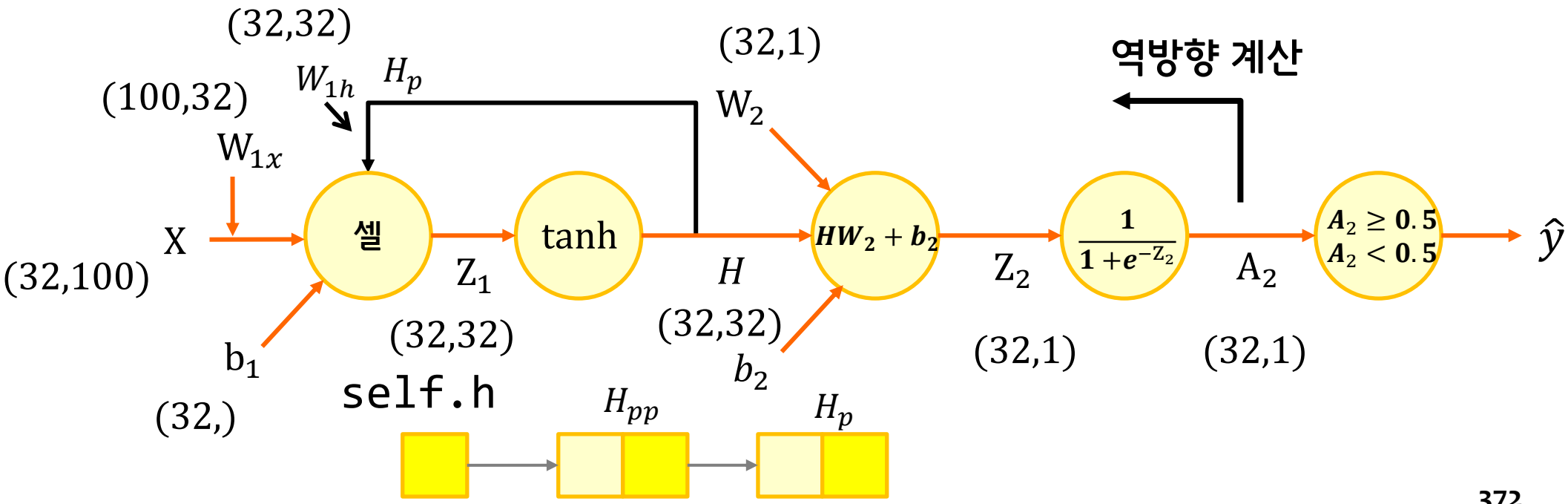
```
        ...
# 배치 차원과 타임 스텝 차원을 바꿉니다.
        seq = np.swapaxes(x, 0, 1)
        for x in seq:
```

(32,100,100) => (100, 32, 100)



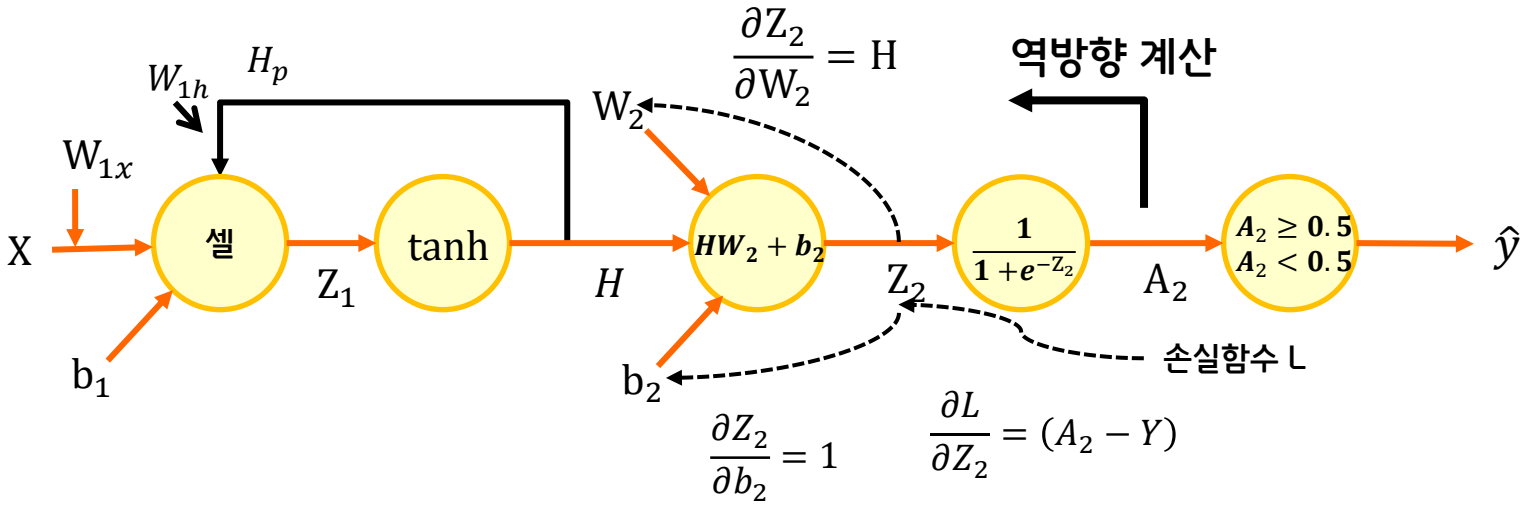
5. 각 샘플의 모든 타임 스텝에 대한 정방향 계산을 수행 한다.

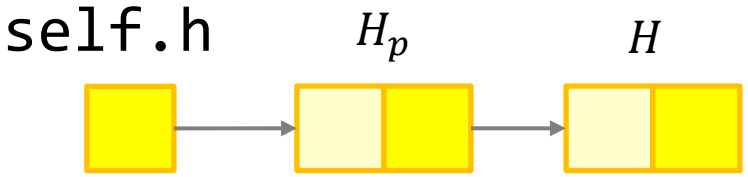
```
...
for x in seq:
        z1 = np.dot(x, self.w1x) + np.dot(self.h[-1], self.w1h) + self.b1
        h = np.tanh(z1)                          # 활성화 함수를 적용합니다.
        self.h.append(h)                         # 역전파를 위해 은닉 상태 저장합니다.
        z2 = np.dot(h, self.w2) + self.b2  # 출력층의 선형 식을 계산합니다.
return z2
```



6. 역방향 계산 구현

```
def backprop(self, x, err):
        m = len(x)       # 샘플 개수

        # 출력층의 가중치와 절편에 대한 그래디언트를 계산합니다.
        w2_grad = np.dot(self.h[-1].T, err) / m
        b2_grad = np.sum(err) / m
```





$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = H^T(A_2 - Y) \qquad (m, n_c)^T(m, 1)$$

$$(n_c, m)(m, 1)$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial b_2} = 1^T(A_2 - Y) \Rightarrow sum(A_2 - Y)$$

6. 역방향 계산 구현

TBPTT

```
def backprop(self, x, err):
        seq = np.swapaxes(x, 0, 1)
        w1h_grad = w1x_grad = b1_grad = 0
        # 셀 직전까지 그래디언트를 계산합니다.
        err_to_cell = np.dot(err, self.w2.T) * (1 - self.h[-1] ** 2)
        # 모든 타임 스텝을 거슬러가면서 그래디언트를 전파합니다.
        for x, h in zip(seq[::-1][:10], self.h[:-1][::-1][:10]):
            w1h_grad += np.dot(h.T, err_to_cell)
            w1x_grad += np.dot(x.T, err_to_cell)
            b1_grad += np.sum(err_to_cell, axis=0)
            # 이전 타임 스텝의 셀 직전까지 그래디언트를 계산합니다.
            err_to_cell = np.dot(err_to_cell, self.w1h) * (1 - h ** 2)

        w1h_grad /= m
        w1x_grad /= m
        b1_grad /= m

        return w1h_grad, w1x_grad, b1_grad, w2_grad, b2_grad
```

```
w1h_grad += np.dot(h.T, err_to_cell)
w1x_grad += np.dot(x.T, err_to_cell)
b1_grad += np.sum(err_to_cell, axis=0)
# 이전 타임 스텝의 셀 직전까지 그래디언트를 계산합니다.
err_to_cell = np.dot(err_to_cell, self.w1h) * (1 - h ** 2)
```

(32, 32) (32, 32)

$$\frac{\partial Z_1}{\partial W_{1h}} = H_p$$

$H_p^T err\_to\_cell$

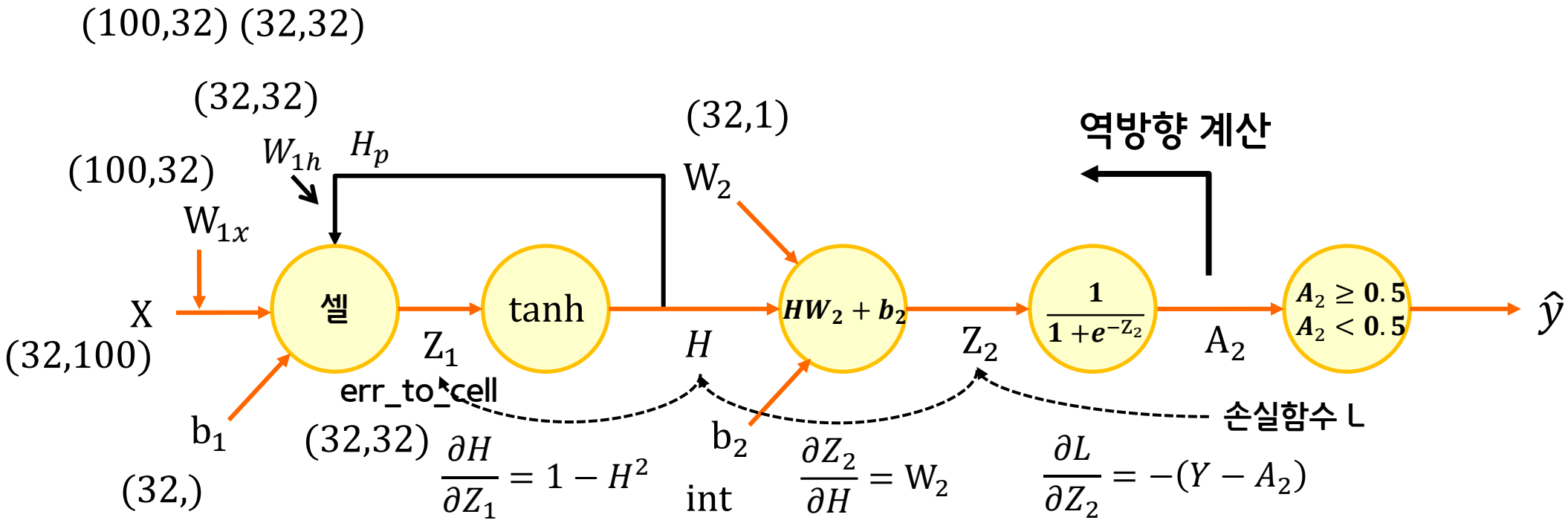
$$\frac{\partial Z_1}{\partial W_{1h}} = H_p + H_{pp}W_{1h}\odot\left(1-H_p^2\right) + H_{ppp}W_{1h}\odot\left(1-H_p^2\right)\odot W_{1h}\odot\left(1-H_{pp}^2\right) + \cdots$$

$$\frac{\partial Z_1}{\partial W_{1x}} = X + X_pW_{1h}\odot\left(1-H_p^2\right) + X_{pp}W_{1h}\odot\left(1-H_p^2\right)\odot W_{1h}\odot\left(1-H_{pp}^2\right) + \cdots$$

```
sum=0
for i in range(10):
        sum = sum + i+1
```

```
 0
(i+1)
(i+1)+i+1
((i+1)+i+1) + i+1
...
```

```
sum = '123'
num = 0

sum[0] =>  '1' =>  49

num = num*10 + sum[i] - '0'

12*10

120
```


(100,32) (32,32)
(32,32)
(32,1)
역방향 계산
(100,32)
W_{1h}
H_p
W_2
W_{1x}
X
(32,100)
셀
tanh
$HW_2 + b_2$
$\frac{1}{1+e^{-Z_2}}$
$A_2 \geq 0.5$
$A_2 < 0.5$
$\hat{y}$
Z_1
H
Z_2
A_2
err_to_cell
손실함수 L
b_1
(32,32)
b_2
(32,)
$\frac{\partial H}{\partial Z_1} = 1 - H^2$
int
$\frac{\partial Z_2}{\partial H} = W_2$
$\frac{\partial L}{\partial Z_2} = -(Y - A_2)$

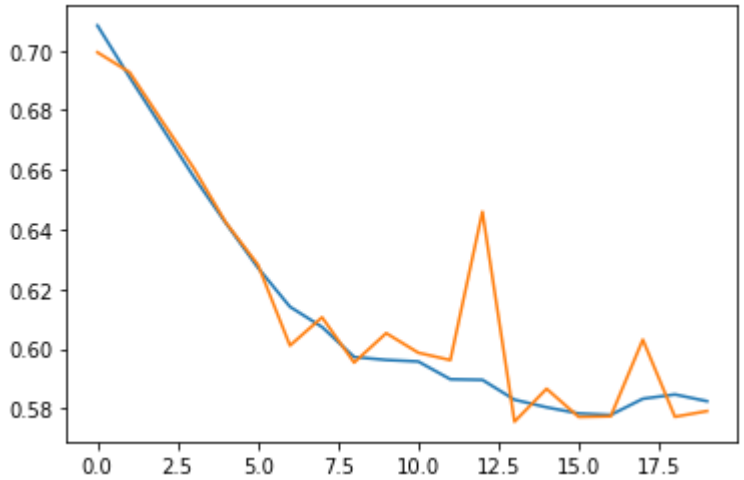
1. 순환 신경망 모델 훈련시키기

```
rn = RecurrentNetwork(n_cells=32, batch_size=32, learning_rate=0.01)

rn.fit(x_train_onehot, y_train, epochs=20, x_val=x_val_onehot, y_val=y_val)
```

```
에포크
0 ..................................................................................
..................................................................................
..................................................................................
..................................................................................
..................................................................................
..................................................................................
.........
에포크
1 ..................................................................................
..................................................................................
..................................................................................
..................................................................................
..................................................................................
..................................................................................
.........
에포크 2
```

2. 훈련, 검증 세트에 대한 손실 그래프 그리기

```
import matplotlib.pyplot as plt
plt.plot(rn.losses)
plt.plot(rn.val_losses)
plt.show()
```



3. 검증 세트 정확도 평가하기

```
rn.score(x_val_onehot, y_val)
```

```
0.7002
```

6. 순환 신경망 이해

6.1 순환 신경망

6.2 순환 신경망의 오차역전파

6.3 순환 신경망 구현

6.4 케라스를 이용한 순환 신경망 구현

1. 순환 신경망에 필요한 클래스 임포트하기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
```

2. 모델 만들기

```
model = Sequential()

model.add(SimpleRNN(32, input_shape=(100, 100)))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 32)                4256  32*100 + 32*32 + 32
_________________________________________________________________
dense (Dense)                (None, 1)                 33    32*1 + 1
=================================================================
Total params: 4,289
Trainable params: 4,289
Non-trainable params: 0
_________________________________________________________________
```

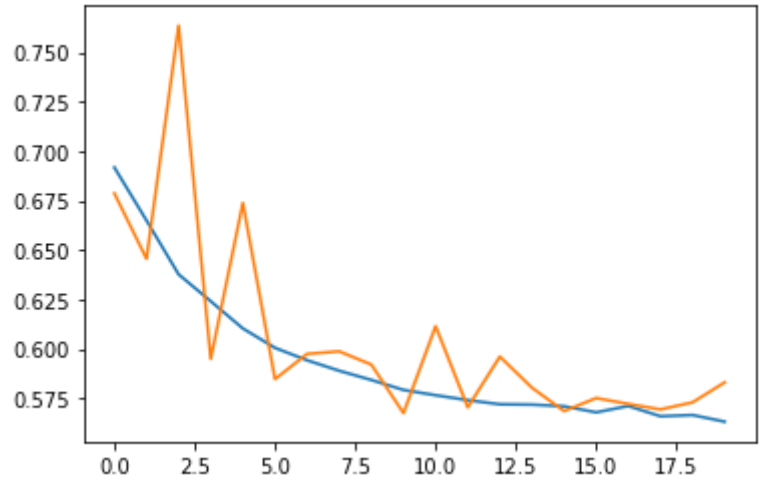
3. 모델 컴파일하고 훈련 시키기

```
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(x_train_onehot, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val_onehot, y_val))
```

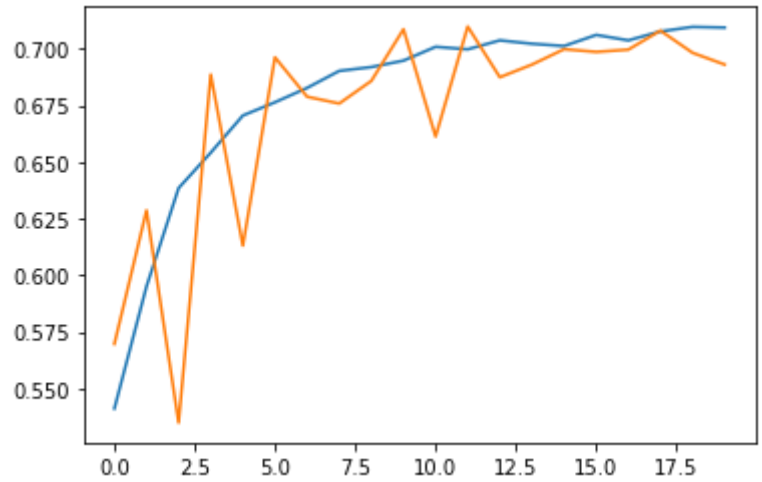
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/20
20000/20000 [==============================] - 16s 816us/sample - loss: 0.7011 - accuracy: 0.5003 -
val_loss: 0.6979 - val_accuracy: 0.4978
Epoch 2/20
20000/20000 [==============================] - 15s 774us/sample - loss: 0.6947 - accuracy: 0.5123 -
val_loss: 0.6961 - val_accuracy: 0.5066
Epoch 3/20
...
Epoch 18/20
20000/20000 [==============================] - 16s 779us/sample - loss: 0.5816 - accuracy: 0.6967 -
val_loss: 0.5731 - val_accuracy: 0.7026
Epoch 19/20
20000/20000 [==============================] - 16s 775us/sample - loss: 0.5784 - accuracy: 0.6974 -
val_loss: 0.5743 - val_accuracy: 0.6986
Epoch 20/20
20000/20000 [==============================] - 16s 776us/sample - loss: 0.5756 - accuracy: 0.7031 -
val_loss: 0.5651 - val_accuracy: 0.7160
```

4. 훈련,검증 세트에 대한 손실 그래프와 정확도 그래프 그리기

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.show()
```



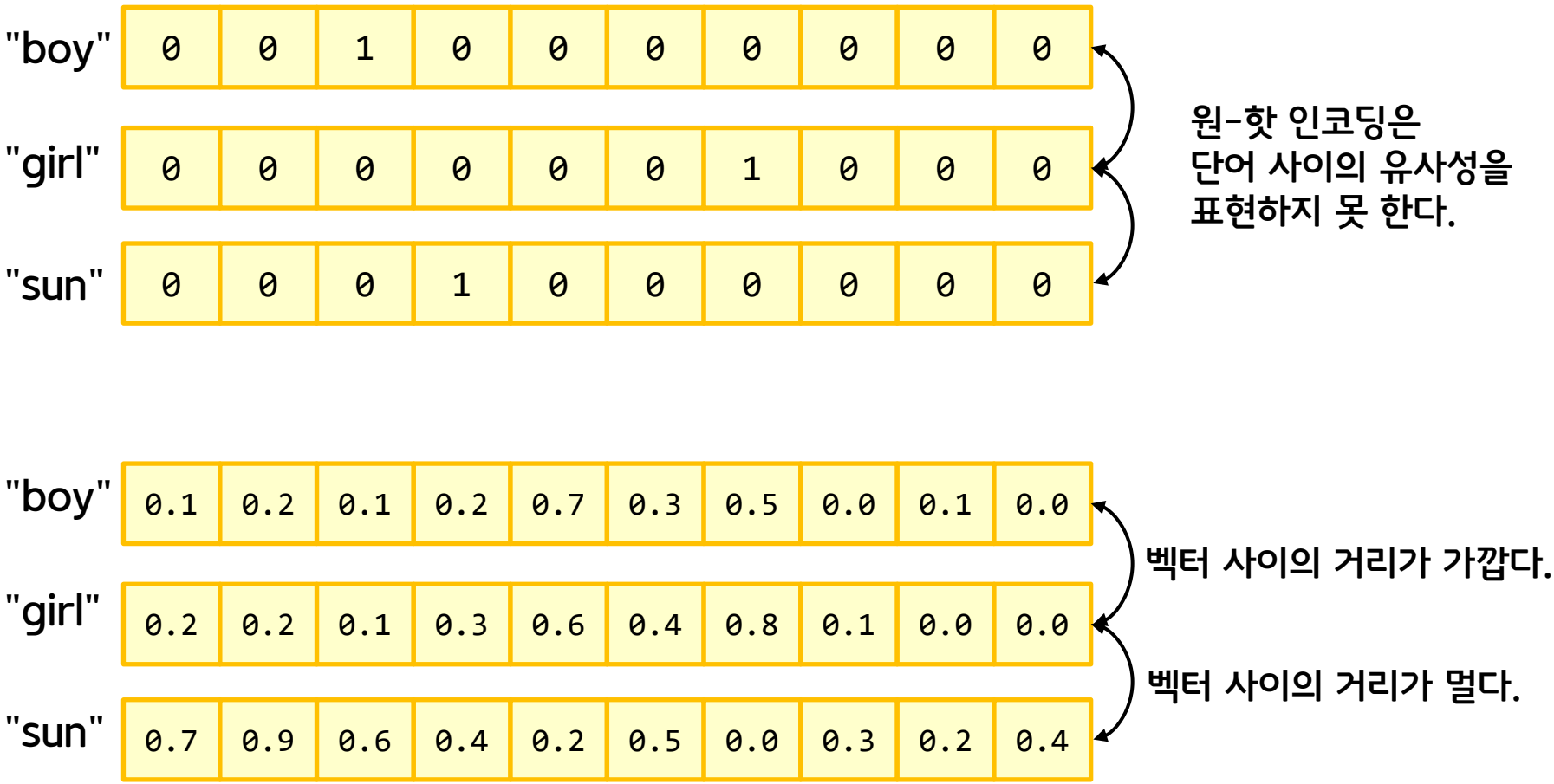

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.show()
```



5. 검증 세트 정확도 평가하기

```
loss, accuracy = model.evaluate(x_val_onehot, y_val, verbose=0)
print(accuracy)
```

```
0.693
```


"boy"
0 0 1 0 0 0 0 0 0 0
"girl"
0 0 0 0 0 0 1 0 0 0
"sun"
0 0 0 1 0 0 0 0 0 0
원-핫 인코딩은
단어 사이의 유사성을
표현하지 못 한다.
"boy"
0.1 0.2 0.1 0.2 0.7 0.3 0.5 0.0 0.1 0.0
"girl"
0.2 0.2 0.1 0.3 0.6 0.4 0.8 0.1 0.0 0.0
"sun"
0.7 0.9 0.6 0.4 0.2 0.5 0.0 0.3 0.2 0.4
벡터 사이의 거리가 가깝다.
벡터 사이의 거리가 멀다.

1. Embedding 클래스 임포트하기

```
from tensorflow.keras.layers import Embedding
```

2. 훈련 데이터 준비하기

```
(x_train_all, y_train_all), (x_test, y_test) = imdb.load_data(skip_top=20, num_words=1000)

for i in range(len(x_train_all)):
    x_train_all[i] = [w for w in x_train_all[i] if w > 2]

x_train = x_train_all[random_index[:20000]]
y_train = y_train_all[random_index[:20000]]
x_val = x_train_all[random_index[20000:]]
y_val = y_train_all[random_index[20000:]]
```

3. 샘플 길이 맞추기

```
maxlen=100
x_train_seq = sequence.pad_sequences(x_train, maxlen=maxlen)
x_val_seq = sequence.pad_sequences(x_val, maxlen=maxlen)
```

4. Embedding 클래스 임포트하기

```
model_ebd = Sequential()

model_ebd.add(Embedding(1000, 32))
model_ebd.add(SimpleRNN(8))
model_ebd.add(Dense(1, activation='sigmoid'))

model_ebd.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          32000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 8)                 328
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 32,337
Trainable params: 32,337
Non-trainable params: 0
_________________________________________________________________
```

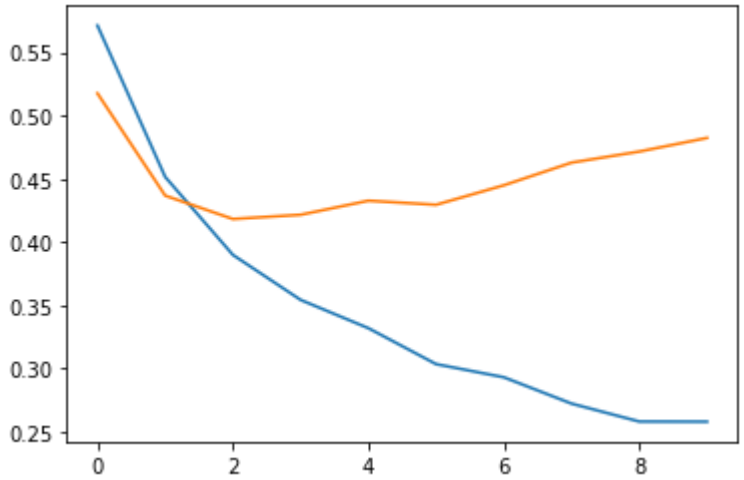
5. 모델 컴파일하고 훈련시키기

```
model_ebd.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model_ebd.fit(x_train_seq, y_train, epochs=10, batch_size=32,
                        validation_data=(x_val_seq, y_val))
```

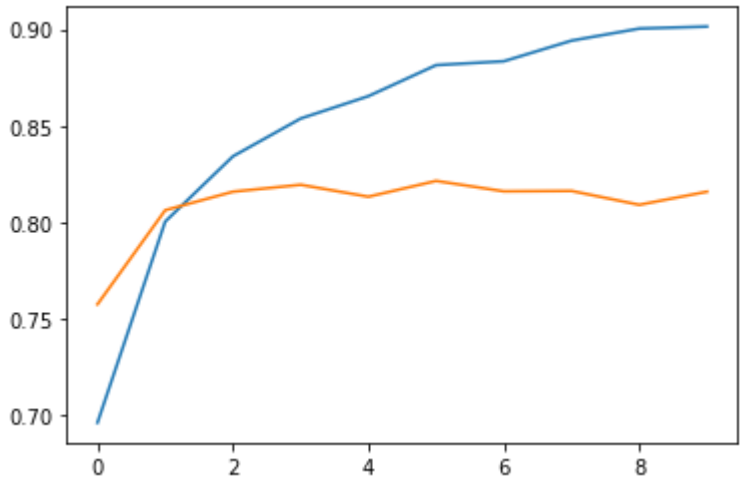
```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 8s 417us/sample - loss: 0.5712 - accuracy: 0.6960 -
val_loss: 0.5176 - val_accuracy: 0.7574
Epoch 2/10
20000/20000 [==============================] - 8s 407us/sample - loss: 0.4515 - accuracy: 0.8003 -
val_loss: 0.4368 - val_accuracy: 0.8062
Epoch 3/10

...

Epoch 9/10
20000/20000 [==============================] - 8s 394us/sample - loss: 0.2581 - accuracy: 0.9004 -
val_loss: 0.4716 - val_accuracy: 0.8090
Epoch 10/10
20000/20000 [==============================] - 8s 393us/sample - loss: 0.2580 - accuracy: 0.9014 -
val_loss: 0.4824 - val_accuracy: 0.8158
```

6. 손실 그래프와 정확도 그래프 그리기

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.show()
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.show()
```



7. 검증 세트 정확도 평가하기

```
loss, accuracy = model_ebd.evaluate(x_val_seq, y_val, verbose=0)
print(accuracy)
```

```
0.8158
```